Information Flow Analysis for Fail-Secure Devices

Andrew Rae Colin Fidge

School of Information Technology and Electrical Engineering,
The University of Queensland, Australia
Email: {arae,cjf}@itee.uq.edu.au

Information security devices must preserve security properties even in the presence of faults. This in turn requires a rigorous evaluation of the system behaviours resulting from component failures, especially how such failures affect information flow. We introduce a compositional method of static analysis for fail-secure behaviour. Our method uses reachability matrices to identify potentially undesirable information flows based on the fault modes of the system's components.

1. INTRODUCTION

The internationally standardised *Common Criteria for Information Security Evaluation* [1] includes 'fail secure' requirement FPT_FLS.1, recommending that the target of evaluation does not violate its security policy in the event of failures. Unfortunately, the corresponding *Common Methodology for Information Technology Security Evaluation* [2] provides little guidance on how the requirement is to be demonstrated or evaluated.

Indeed, evaluating the fault behaviour of a device is extremely challenging. Not only must the consequences of every possible failure of every component within the device be examined, but consideration must be given to the consequences of simultaneous complicit faults, involving several components. Furthermore, component faults may be independent, as per the failure modes of specific pieces of equipment, or related, due to some overall design flaw.

In this paper we show how information-flow analysis can be used to isolate those fault modes that require close evaluation. This is done by representing input-output dependencies in a tabular format, where each cell can hold several different fault modes, and by then using these tables to calculate end-to-end information flow in different modes. Both top-down and bottom-up analyses are supported by allowing for system-wide and individual-component failures, respectively.

2. PREVIOUS WORK

Our research takes a graph-theoretic approach to evaluating failure modes. The connectivity of components within a device is represented as a table on which reachability analysis is performed to detect undesirable information flow. This builds on well-established techniques for analysing information flow through computer programs [3]. However, unlike our work, previous 'information flow relations' do not take system modes or states into account.

Earlier research into formal evaluation of security devices also used graphs and tables to model the dependencies between 'monitored variables' (inputs) and 'controlled variables' (outputs) [4]. Such graphs were used to assess the 'complexity' of a design and to look for circular dependencies. However, unlike our work, these graphs have not been used to evaluate system faults.

Our method is specifically focussed at identifying violations of *confidentiality*, that is, illicit flow of information through a system. This is distinguished from previous information-flow techniques [3], which focus on *integrity*, that is, the correctness of a program.

3. FAULT-MODE DEPENDENCY TABLES

In this section we define our fault-mode dependency tables and their compositional behaviour. We firstly introduce the concept for a system with just one mode, then extend it for system-wide, related failures and then for local, independent component failures.

3.1. Modelling Information Flow

The information-flow behaviour of a system component can be considered as a transfer function in which the component's outputs depend on the values of its inputs. This function may vary according to the component's *state* or *mode*. For a component with multiple inputs and outputs, the value of each output may depend on a subset of the inputs only.

As a running example, consider the schematic layout shown in Figure 1. Component X has inputs A and B, and output C. Component Y has inputs C and D, and outputs E and F. In considering the combined behaviour of these two components we say that the overall system has inputs A, B and D, outputs E and F,

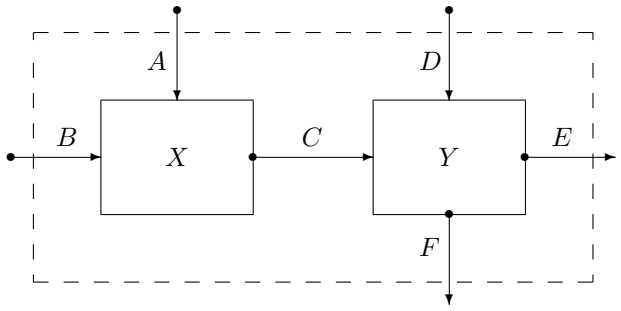

FIGURE 1. Example component layout

X	C
A	1
B	0

FIGURE 2. Binary dependency table for component X

Y	E	F
C	1	0
D	1	1

FIGURE 3. Binary dependency table for component Y

and an *internal* connection C. We assume that each 'connection' is an arc with two endpoints; 'forking' or 'joining' must be modelled with an explicit component at the point of intersection, rather than by having a branching connection.

When evaluating the security of such a system we are concerned with connectivity from inputs to outputs, especially when the input resides on the high-security side of the device's environment and the output on the low-security side. However, merely treating Figure 1 as a graph and performing reachability analysis is unhelpful; this would tell us only that both outputs E and F *may* depend on any or all inputs A, B and D. Instead we need to reduce the number of cases to be evaluated by considering the specific ways in which components X and Y connect their inputs to their outputs. Furthermore, the task is made harder by the fact that these components may establish different connections in different modes. Therefore, our goal is to undertake a form of reachability analysis that takes into account the mode-specific way that components make connections.

Information-flow analysis for high-level language software has shown how dependencies among variables in a program can be calculated using simple matrix manipulations [3, §10.3]. We adopt this principle as the basis for our *fault-mode dependency tables* for analysis of security device schematics by extending the basic concept with a notion of fault modes.

3.2. Composing Information Flow

As an introductory exercise, we begin by explaining how to combine the dependency tables associated with two components, assuming that there is just one system mode. In this situation binary tables are sufficient. The approach builds on concepts for information-flow analysis between program variables in high-level language software [3, §10.3]. However, where software analysis assumes that all variables act as both inputs and outputs, here we maintain the distinction between inputs, outputs and internal connections.

The first step is to identify the input-output relationships for each component, based on the information available to the security evaluator, e.g., via the manufacturer's data sheet for the component. To analyse this information it is then encoded in the form of dependency tables. The general form of such a table is as follows:

component	outputs
inputs	

For the given component, the table's rows list its inputs and the columns list its outputs. The contents of each cell determine whether or not a particular output is dependent on the corresponding input. (In program analysis [3], effort is required to determine whether a software variable is used as an input, output or both to a block of code. In our application the role of each connection is immediately obvious from the schematic diagram.) In the case where there is only one mode, the cells can be filled with binary values. Let '1' in a cell mean that the output depends on the corresponding input, and '0' mean that the output is independent of the input.

As an example, a possible dependency table for component X in Figure 1 is shown in Figure 2. It tells us that component X's output C depends on its input A, but not on unused input B. Similarly, the table in Figure 3 models the fact that component Y's output E depends on both of its inputs, but output F depends on input D only.

Our goal now is to combine these two tables to produce the dependency relationships for the whole system. It has been shown elsewhere that matrix mutiplication provides a satisfactory way of composing information flows [3, §10.3]. However, recall that to multiply two matrices the number of columns in the left-hand matrix must equal the number of rows in the right-hand one. Therefore, we must first 'fill in' the (human-friendly) dependency tables in Figures 2 and 3 to produce equivalent (machine-manipulable) matrices.

We do this by adding rows and columns for every connection in the system, so that the rows and columns both list all inputs, outputs and internal connections. (In effect, the tables above are sparse matrices which we need to complete.) The new cells thus created are filled in as follows:

X	A	B	C	D	E	F
A	1	0	1	0	0	0
B	0	1	0	0	0	0
C	0	0	1	0	0	0
D	0	0	0	1	0	0
E	0	0	0	0	1	0
F	0	0	0	0	0	1

FIGURE 4. Binary matrix for component X

Y	A	B	C	D	E	F
A	1	0	0	0	0	0
B	0	1	0	0	0	0
C	0	0	1	0	1	0
D	0	0	0	1	1	1
E	0	0	0	0	1	0
F	0	0	0	0	0	1

FIGURE 5. Binary matrix for component Y

- cells on the diagonal of the matrix are filled with '1', because every connection 'depends on' itself; and
- all other newly created cells are filled with '0', because we don't want to introduce any dependencies not identified by the security evaluator.

Thus, the two dependency tables in Figures 2 and 3 can be transformed into the matrices of Figures 4 and 5.

Each of these matrices models the connections through a single component. A matrix modelling the set of all single-component connections can then be found by 'adding' the two matrices. For two binary matrices X and Y of equal dimensions, let their sum $X+Y$ be a matrix such that the cell in the ith row and jth column is defined as follows.

$$(X+Y)_{i,j} \stackrel{\text{def}}{=} \max(X_{i,j}, Y_{i,j})$$

However, the resulting matrix still only defines dependencies through individual components, i.e., it shows connections between inputs and outputs that can be reached in a single step. To tell if we can reach a given output from a given input by traversing both components X and Y, we need to take the 'square' of the combined matrix. For a given binary matrix Z, of dimension $N \times N$, let its square Z^2 be a matrix such that the cell in the ith row and jth column is defined as follows.

$$Z^2_{i,j} \stackrel{\text{def}}{=} \max_{1 \leqslant k \leqslant N} (Z_{i,k} * Z_{k,j})$$

The outer 'max' operator iterates over all corresponding cells k in row i and column j and thus merges all possible connections between input i and output j. (We use 'max' rather than matrix multiplication's usual '$\sum$' operation so that the outcome remains binary.) The inner '$*$' operator returns '1' only if both cells contain

$(X+Y)^2$	A	B	C	D	E	F
A	1	0	1	0	1	0
B	0	1	0	0	0	0
C	0	0	1	0	1	0
D	0	0	0	1	1	1
E	0	0	0	0	1	0
F	0	0	0	0	0	1

FIGURE 6. Composition of binary matrices for components X and Y

$(X+Y)^2$	E	F
A	1	0
B	0	0
D	1	1

FIGURE 7. Composition of binary dependency tables for components X and Y

'1', i.e., if input i is connected to internal connection k and k is connected to output j.

Applying these two operators to the matrices in Figures 4 and 5 produces the matrix in Figure 6. This outcome precisely matches our intuitions about the combined behaviour of these two components. The diagonal cells show that every connection affects itself, as before. All the other assumed dependencies, such as output C's dependency on input A and output F's dependency on input D, are transferred from the individual components. Most importantly, however, the combined matrix reveals that output E may depend on input A. This is because output E depends on connection C, according to component Y's dependency table, and connection C depends on input A, according to component X's dependency table.

Finally, the combined binary matrix can be turned back into a more readable dependency table for the overall system by:

- deleting rows labelled with outputs and internal connections; and
- deleting columns labelled with inputs and internal connections.

This hides internal connections and eliminates cells that are usually irrelevant because they map outputs to inputs, inputs to inputs, or outputs to outputs.

The final dependency table for our complete system is thus shown in Figure 7. It tells us that, in the overall system, output E depends on inputs A and D, output F depends on input D, and no outputs depend on unused input B, as expected.

Since we have only two components in this example, squaring the matrix was sufficient. More generally, though, there are three possible ways of deciding how many times to multiply the matrices, in order to ensure that all transitive relationships from inputs to outputs are calculated.

- The most obvious approach is to just keep multiplying until the matrix stops changing (which it will, even if there are loops in the graph, because the dependencies are a static property of the graph).
- A simpler, but sub-optimal, approach is to multiply as many times as there are connections in the graph (which is guaranteed to complete the calculation for the longest possible path, i.e., when the graph is linear).
- An approach that is optimal in terms of the number of multiplications is to multiply as many times as the length of the longest path through the graph that doesn't revisit connections (but this requires a separate graph analysis to determine this number).

The example we have provided in this section shows a simple component layout. However, the technique generalises readily to more complex layouts, such as where a single input feeds multiple components, or where there are multiple connections between the same components.

3.3. System-Wide Fault Modes

Having seen how dependencies can be calculated for a system with only one mode, we now consider how the concept can be extended to handle a system with several fault modes. In this section we consider 'system-wide' failures since this allows the evaluator to model situations where faults are related, e.g., due to the close physical proximity of several components to the same potential source of damage. This supports a top-down form of fault analysis.

Consider again the system of Figure 1, and assume that the security evaluator's analysis of the device's construction has identified the following fault modes:

0: no faults

1: connection D broken

2: component X directly links input B to connection C

3: connection D broken and input B linked to connection C

Note that while all fault *modes* are assumed to be mutually-exclusive, i.e., the system is always in exactly one of the four modes above, no such assumption is made about the faults themselves. Thus, fault mode 3 models the situation where the faults of modes 1 and 2 combine.

To handle different fault modes, we extend the dependency tables of Section 3.2 so that the cells hold not binary values, but sets of modes. For instance, Figures 8 and 9 show the dependency tables for components X and Y, respectively, listing the modes in which dependencies exist.

X	C
A	$\{0,1,2,3\}$
B	$\{2,3\}$

FIGURE 8. Dependency table for component X with system-wide failures

Y	E	F
C	$\{0,1,2,3\}$	$\emptyset$
D	$\{0,2\}$	$\{0,2\}$

FIGURE 9. Dependency table for component Y with system-wide failures

In this case component X's output C depends on input A in all modes. However, output C depends on input B only in the two fault modes where a short circuit has been created between these connections. Component Y's output E depends on its input C in all modes. However, outputs E and F depend on input D only in those modes where input D is not broken.

The dependency tables can then be transformed into complete matrices as before, except that:

- the diagonal of the matrix contains the set of all modes; and
- newly created cells are filled with the empty set $\emptyset$.

Thus, the full matrices derived from the dependency tables in Figures 8 and 9 are shown in Figures 10 and 11, respectively.

In this situation, matrices X and Y are added by taking the union of the corresponding sets of fault modes.

$$(X+Y)_{i,j} \quad \stackrel{\text{def}}{=} \quad X_{i,j} \cup Y_{i,j}$$

The resulting $N \times N$ matrix Z can then be 'squared' by taking the intersection of the corresponding sets of failure modes.

$$Z^2_{i,j} \quad \stackrel{\text{def}}{=} \quad \bigcup_{1 \leqslant k \leqslant N} (Z_{i,k} \cap Z_{k,j})$$

The outer '$\cup$' operator iterates over corresponding cells k in row i and column j and thus merges all connections between input i and output j via k. The inner '$\cap$' operator works on individual sets of fault modes and models the fact that a connection exists between input i and output j, via internal connection k, only if information flows from i to k and from k to j in the *same* system-wide fault mode.

Thus the matrices in Figures 10 and 11 can be combined to produce the one shown in Figure 12. Redundant rows and columns can then be eliminated as explained in Section 3.2 to give the final dependency table shown in Figure 13.

Again, the result matches our intuition for this system. Output E depends on input A in all modes,

X	A	B	C	D	E	F
A	$\{0,1,2,3\}$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\emptyset$	$\emptyset$
B	$\emptyset$	$\{0,1,2,3\}$	$\{2,3\}$	$\emptyset$	$\emptyset$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\emptyset$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\emptyset$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$

FIGURE 10. System-wide fault-mode matrix for component X

Y	A	B	C	D	E	F
A	$\{0,1,2,3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
B	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\{0,2\}$	$\{0,2\}$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$

FIGURE 11. System-wide fault-mode matrix for component Y

$(X+Y)^2$	E	F
A	$\{0,1,2,3\}$	$\emptyset$
B	$\{2,3\}$	$\emptyset$
D	$\{0,2\}$	$\{0,2\}$

FIGURE 13. Composition of dependency tables for components X and Y with system-wide fault modes

X	C
A	$\{\{0\},\{2\}\}$
B	$\{\{2\}\}$

FIGURE 14. Dependency table for component X with local fault modes

and it depends on input D in all modes where D is not broken. However, output E depends on input B only when there is a short circuit between B and internal connection C. Output F depends on input D only, and then only in modes where D is not broken.

An advantage of using system-wide failure modes is that it facilitates analysis of fault modes which encompass more than one component. In the example above, for instance, the combination of a fault in one of component Y's inputs and an internal fault in component X was explicitly represented.

3.4. Component Fault Modes

In the previous section we assumed that fault modes were identified for the overall system because they may be related. A complementary approach is to identify fault modes for individual components independently, e.g., using the components' data sheets, and then compose them to produce combined fault modes. This allows for a bottom-up form of failure analysis.

For instance, again consider the schematic layout of Figure 1, and assume that the evaluator has separately identified possible failures of components X and Y. The potential fault modes defined for component X are as follows:

$\{0\}$: no faults in component X

$\{1\}$: connection A broken

$\{2\}$: output C depends on input B as well as input A

Then the dependency table for component X refers to these three modes only, as shown in Figure 14. In the case of independent fault modes we use *sets* to model the modes in which dependencies exist. (This is necessary to support later composition.) Thus, Figure 14 tells us that output C depends on input A only if A is not broken, whereas C depends on input B only if component X has a short circuit. Again, although the fault *modes* for individual components are assumed to be mutually-exclusive, i.e., the component is assumed to always be in exactly one of the modes above, each mode may model several simultaneous failures within the component.

Y	E	F
C	$\{\{3\},\{4\}\}$	$\emptyset$
D	$\{\{3\}\}$	$\{\{3\}\}$

FIGURE 15. Dependency table for component Y with local fault modes

Similarly, assume that the fault modes for component Y are the following:

$\{3\}$: no faults in component Y

$\{4\}$: connection D broken

The corresponding dependency table is then shown in Figure 15. It tells us that output E depends on

$(X+Y)^2$	A	B	C	D	E	F
A	$\{0,1,2,3\}$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
B	$\emptyset$	$\{0,1,2,3\}$	$\{2,3\}$	$\emptyset$	$\{2,3\}$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\{0,2\}$	$\{0,2\}$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{0,1,2,3\}$

FIGURE 12. Composition of system-wide fault-mode matrices for components X and Y

input C in all modes, whereas output F never depends on C. Also, both outputs E and F depend on input D provided that D is not broken.

As before, we need to turn these tables (sparse matrices) into complete matrices. In this case the individual matrices are completed by:

- filling the diagonal cells with the set of all singleton fault-mode sets for the component; and
- filling all other newly-created cells with the empty set $\emptyset$.

The resulting matrix for component X is shown in Figure 16, and that for component Y in Figure 17.

Both of these components are atomic, so the cells of their fault-mode dependency tables were populated by sets of singleton fault-mode sets. More generally, however, we must consider how to represent the fault modes of composite components consisting of two or more atomic components with independent faults. For an atomic component $\mathcal{C}_i$, let $\mathcal{F}_i$ denote the set of its possible fault modes. In the example above, $\mathcal{F}_X$ is $\{\{0\},\{1\},\{2\}\}$ and $\mathcal{F}_Y$ is $\{\{3\},\{4\}\}$. Then the set $\mathcal{F}$ of possible fault modes for a composite component $\mathcal{C}$, comprising atomic components $\mathcal{C}_1,\ldots,\mathcal{C}_n$, is composed of sets of atomic faults, one for each atomic component.

$$\mathcal{F} \stackrel{\text{def}}{=} \{f_1 \cup \cdots \cup f_n \mid f_1 \in \mathcal{F}_1 \wedge \cdots \wedge f_n \in \mathcal{F}_n\}$$

In the example above, the set of all possible fault modes for the composition of components X and Y is thus $\{\{0,3\},\{1,3\},\{2,3\},\{0,4\},\{1,4\},\{2,4\}\}$.

To 'add' independent fault-mode matrices, we therefore need to define an addition formula that takes this composition of individual fault modes into account. For two components X and Y, with possible fault modes $\mathcal{F}_X$ and $\mathcal{F}_Y$, respectively, the cell in the ith row and jth column of the combined matrix is defined as shown below.

$$(X+Y)_{i,j} \stackrel{\text{def}}{=} \begin{cases} \{x \cup y \mid x \in X_{i,j} \wedge y \in \mathcal{F}_Y\}, & Y_{i,j} = \emptyset \\ \{x \cup y \mid x \in \mathcal{F}_X \wedge y \in Y_{i,j}\}, & X_{i,j} = \emptyset \\ \{x \cup y \mid x \in X_{i,j} \wedge y \in Y_{i,j}\}, & \text{otherwise} \end{cases}$$

There are three cases. In the first case component X allows information flow from input i to output j in certain modes, but component Y has no modes that link i to j. Since the components' fault modes are independent, the modes in which the composite component '$X+Y$' connects i to j are thus those composite modes constructed from component X's relevant modes in $X_{i,j}$ and any possible modes in $\mathcal{F}_Y$ for component Y. Vice versa for the second case where component Y connects i to j, but X does not. The third case handles the cells on the diagonal and creates the cross product of all fault modes from both components. (By precluding forking and branching connections in Section 3.1 we ensured that only one of two components X and Y can link a given input i to an output j, provided that X and Y contain no subcomponents in common. In other words, at least one cell $X_{i,j}$ or $Y_{i,j}$ will always be the empty set, for distinct indices i and j.)

Provided that all matrices obey the compositional principles described above, then 'squaring' matrices containing well-formed sets of independent fault modes works in exactly the same way as for system-wide fault modes.

$$Z^2_{i,j} \stackrel{\text{def}}{=} \bigcup_{1 \leqslant k \leqslant N} (Z_{i,k} \cap Z_{k,j})$$

In this case the inner '$\cap$' operator takes the intersection of sets of sets of individual faults, and thus models the fact that information flows from i to j via k when the two components are both in compatible modes.

In the example, the matrices in Figures 16 and 17 are therefore composed to yield the one in Figure 18. For brevity, let '$*$' denote set $\{\{0,3\},\{1,3\},\{2,3\},\{0,4\},\{1,4\},\{2,4\}\}$.

Irrelevant rows and columns can then be removed as before, to produce the final composite dependency table shown in Figure 19. Once more, we have a meaningful result. The table tells us that output E depends on input A in any composite mode where there are no faults in component X, or component X has a short circuit between input B and output C. By contrast, output E depends on input B only in modes where the short circuit exists. Output F never depends on input A or B. Outputs E and F depend on input D in all modes where component Y's connection to D is not broken.

X	A	B	C	D	E	F
A	$\{\{0\},\{1\},\{2\}\}$	$\emptyset$	$\{\{0\},\{2\}\}$	$\emptyset$	$\emptyset$	$\emptyset$
B	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$	$\{\{2\}\}$	$\emptyset$	$\emptyset$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$	$\emptyset$	$\emptyset$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$	$\emptyset$	$\emptyset$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$

FIGURE 16. Local fault-mode matrix for component X

Y	A	B	C	D	E	F
A	$\{\{3\},\{4\}\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
B	$\emptyset$	$\{\{3\},\{4\}\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
C	$\emptyset$	$\emptyset$	$\{\{3\},\{4\}\}$	$\emptyset$	$\{\{3\},\{4\}\}$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{3\},\{4\}\}$	$\{\{3\}\}$	$\{\{3\}\}$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{3\},\{4\}\}$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\{3\},\{4\}\}$

FIGURE 17. Local fault-mode matrix for component Y

4. CASE STUDY: A KEYBOARD SWITCH

As a more substantial case study we consider a 'Keyboard Switch', which itself may be part of a Multi-Computer (or Keyboard-Video-Mouse) Switch. Such devices are used to allow peripheral equipment to be shared between computers residing in different security domains. In this example the Keyboard Switch device is intended to connect a single keyboard to either of two computers, depending on the state of a toggle switch.

Figure 20 shows the Keyboard Switch's schematic diagram. Its components are the keyboard, a toggle switching component, and two microprocessors, PIC1 and PIC2. Typically the microprocessors must perform two major functions. Firstly, they must allow both computers to complete their boot sequences in the belief that they each have a dedicated keyboard attached. Then, in normal operating mode, while one microprocessor is forwarding keystrokes from the keyboard to the appropriate computer, the other must convince the 'disconnected' computer that it still has a keyboard attached. The 'toggle switcher' controls the behaviour of the two microprocessors, depending on the position of a physical switch controlled by the operator. (To avoid the computers reacting to having their keyboard 'unplugged,' the switching component does not actually disconnect and reconnect the physical lines.)

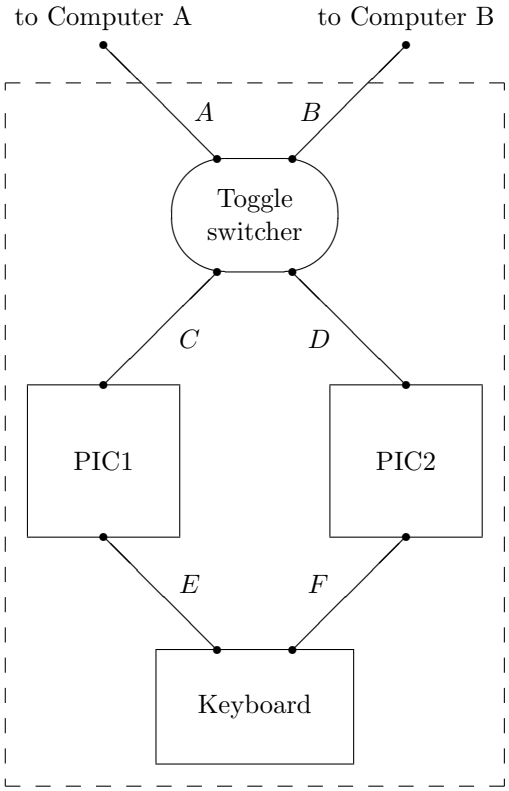

FIGURE 20. The Keyboard Switch

Although the primary information flow through the device consists of keystrokes going from the keyboard to either of the computers, the computers may try to send signals back to the keyboard to, for instance, control LEDs that inform the operator of the Caps-Lock status, etc. For this reason, all of the connections shown in Figure 20 are potentially bidirectional. (For simplicity here we also assume that data passes through the switching component although, as we shall see, such a design does not provide a clear separation of security concerns.) The obvious security issue, therefore, is that a failure in the Keyboard Switch could allow sensitive information to flow from one computer to the other.

In this case we will undertake a bottom-up evaluation, using individual component fault modes. The toggle switching component has three modes:

$(X+Y)^2$	A	B	C	D	E	F
A	$*$	$\emptyset$	$\{\{0,3\},\{0,4\},$ $\{2,3\},\{2,4\}\}$	$\emptyset$	$\{\{0,3\},\{0,4\},$ $\{2,3\},\{2,4\}\}$	$\emptyset$
B	$\emptyset$	$*$	$\{\{2,3\},\{2,4\}\}$	$\emptyset$	$\{\{2,3\},\{2,4\}\}$	$\emptyset$
C	$\emptyset$	$\emptyset$	$*$	$\emptyset$	$*$	$\emptyset$
D	$\emptyset$	$\emptyset$	$\emptyset$	$*$	$\{\{0,3\},\{1,3\},\{2,3\}\}$	$\{\{0,3\},\{1,3\},\{2,3\}\}$
E	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	$\emptyset$
F	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$*$

FIGURE 18. Composition of local fault-mode matrices for components X and Y

$(X+Y)^2$	E	F
A	$\{\{0,3\},\{0,4\},\{2,3\},\{2,4\}\}$	$\emptyset$
B	$\{\{2,3\},\{2,4\}\}$	$\emptyset$
D	$\{\{0,3\},\{1,3\},\{2,3\}\}$	$\{\{0,3\},\{1,3\},\{2,3\}\}$

FIGURE 19. Composition of dependency tables for components X and Y with local fault modes

PIC1	C	E
C	$\{\{4\},\{5\}\}$	$\{\{5\}\}$
E	$\{\{4\},\{5\}\}$	$\{\{4\},\{5\}\}$

FIGURE 22. Dependency table for PIC1

PIC2	D	F
D	$\{\{6\},\{7\}\}$	$\{\{6\}\}$
F	$\{\{6\},\{7\}\}$	$\{\{6\},\{7\}\}$

FIGURE 23. Dependency table for PIC2

Keyboard	E	F
E	$\{\{8\},\{9\}\}$	$\{\{9\}\}$
F	$\{\{9\}\}$	$\{\{8\},\{9\}\}$

FIGURE 24. Dependency table for the keyboard

$\{0\}$: switcher working correctly

$\{1\}$: switcher with connections A and B shorted together

$\{2\}$: switcher cross wired, so that A connects to D and B connects to C

The dependency table for the toggle switcher is shown in Figure 21. Since all four connections are bidirectional, they all appear in both the input rows and output columns. Normally the switch will either connect A and C, or B and D, as shown by the four occurrences of 'fault' mode $\{0\}$, ignoring those on the diagonal. In fault mode $\{1\}$, however, B can additionally depend on A and vice versa. In fault mode $\{2\}$, the switch will either connect A and D or B and C.

Microprocessor PIC1 is programmed to securely communicate between the keyboard and computer A, as directed by the toggle switcher. When A is the 'connected' computer, PIC1 forwards keystrokes received from connection E to connection C. When A is 'disconnected', PIC1 discards keystrokes received. In either situation, PIC1 accepts and responds to signals from computer A to preserve the computer's belief that it has a dedicated keyboard attached. To localise the flow of potentially confidential data, PIC1 is programmed not to forward information from computer A to the keyboard. Here, however, we assume that a failure in PIC1 may allow such information flow to occur.

$\{4\}$: PIC1 working normally

$\{5\}$: PIC1 leaking information to keyboard

Figure 22 shows the dependency table for microprocessor PIC1. Normally, in mode $\{4\}$, it allows information to flow from E to C only. In fault mode $\{5\}$, however, information can also flow from C to E.

The fault modes for PIC2 are analogous and its dependency table is shown in Figure 23.

$\{6\}$: PIC2 working normally

$\{7\}$: PIC2 leaking information to keyboard

The keyboard is configured to simply send each keystroke along both channels E and F (following which the 'connected' microprocessor will forward them to the appropriate computer, and the disconnected microprocessor will ignore them). Since connections E and F are effectively wired together at the keyboard, we assume that under certain circumstances it is possible for signals received on one to be echoed on the other.

$\{8\}$: keyboard working securely

$\{9\}$: keyboard exchanging information between its connections

Figure 24 shows the dependency table for the keyboard. Normally, in mode $\{8\}$, neither connection depends on

Switcher	A	B	C	D
A	$\{\{0\},\{1\},\{2\}\}$	$\{\{1\}\}$	$\{\{0\},\{1\}\}$	$\{\{2\}\}$
B	$\{\{1\}\}$	$\{\{0\},\{1\},\{2\}\}$	$\{\{2\}\}$	$\{\{0\},\{1\}\}$
C	$\{\{0\},\{1\}\}$	$\{\{2\}\}$	$\{\{0\},\{1\},\{2\}\}$	$\emptyset$
D	$\{\{2\}\}$	$\{\{0\},\{1\}\}$	$\emptyset$	$\{\{0\},\{1\},\{2\}\}$

FIGURE 21. Dependency table for toggle switcher

the other, but in fault mode $\{9\}$ connection E depends on F and vice versa.

Given these tables it then possible to apply the procedure described in Section 3.4 to calculate the dependencies created by composite failure modes. The tables can be combined two at a time. The order in which this is done is not significant because the table composition definitions are associative.

However, undertaking this procedure reveals that the size of the tables grows dramatically, due to the large number of composite fault modes generated. The first composition generates fault modes each containing two local faults, the second generates fault mode 'triples' and so on. Indeed, for a system comprising x components, each of which has y modes on average, there are y^x distinct global fault modes to consider! Even the small example described above can produce up to $3 * 2 * 2 * 2 = 24$ distinct fault modes and, since there are 36 cells in the dependency table, there can be up to $24 * 36 = 864$ fault mode entries.

Thus, the final dependency table for the whole Keyboard Switch is very complicated. Fortunately, however, only a few of the cells are of importance. In particular, we are concerned by those that show information flow between A and B, because these reveal modes in which information may flow from one security domain to the other. For instance, the cell showing when connection B's value depends on that of connection A contains the following set.

$$\{\{1,4,6,8\},\{1,4,7,8\},\\ \{1,5,6,8\},\{1,5,7,8\},\\ \{1,4,6,9\},\{1,4,7,9\},\\ \{1,5,6,9\},\{1,5,7,9\},\\ \{0,5,6,9\},\{0,5,7,9\},\\ \{2,4,6,9\},\{2,5,6,9\}\}$$

This tells us that there are twelve fault modes in which this undesirable information flow *may* potentially occur, according to our model of the Keyboard Switch's possible failures. The twelve modes identified can be partitioned into three distinct groups.

- The first eight modes, from $\{1,4,6,8\}$ to $\{1,5,7,9\}$, all include component failure 1 in which the toggle switcher has directly connected A and B. This outcome reveals that the switching component is a single point of failure in this design, and explains why combining switching and data transfer functions in a single component is considered poor security design.
- In modes $\{0,5,6,9\}$ and $\{0,5,7,9\}$ microprocessor PIC1 is incorrectly sending data to the keyboard and the keyboard is incorrectly exchanging data between its two connections. (The second mode additionally includes the failure of microprocessor PIC2.) This outcome tells us that there is a potential security violation due to the combined effect of two independent failures in these separate components.
- The final two modes, $\{2,4,6,9\}$ and $\{2,5,6,9\}$, both include the situation where the toggle switcher is cross wired and the keyboard is exchanging data between its connections. The apparent security violation is that data from computer A could flow through PIC2 (due to the cross wiring), through the keyboard (due to the echoing of signals), through PIC1, and out to computer B (again via the cross-wired toggle switcher). However, this result is in fact a 'false positive.' This information-flow scenario would not occur in practice because only one of the two microprocessors is 'connected' and forwarding data at any given moment.

Once the final dependency table has been generated it is up to the security evaluator to interpret the results appropriately. For instance, the first group of modes above revealed that the 'toggle switcher' is a single point of failure in the Keyboard Switch device, so the evaluator would be obliged to use the manufacturer's measured failure rates for this component to determine whether or not this risk is acceptable. Similarly, the evaluator must decide whether the pair of complicit failures in the second group is sufficiently likely to warrant concern. For the third group, the evaluator needs to explain why this potential data-flow scenario is not possible in practice. (Generation of this false positive is not an error in our calculations, but is a limitation of the particular failure model of the Keyboard Switch used, which did not fully represent all of the normal operating states of the microprocessors.)

5. DISCUSSION

Ideally all system components should be treated uniformly under our methodology, regardless of whether they are implemented as hardware or software. Thus, programmable components would be represented as a composition of a hardware component and a software component. The hardware component includes

physical inputs and outputs, as well as logical variables which can be treated as 'connections' to the software. However, production of a dependency table for computer software presents a challenge outside the scope of this paper. Readers are referred to Barnes [3] for a method of producing mode-independent dependency tables for software. However, determining meaningful fault modes for computer programs is an open problem in the safety-critical systems literature [5, 6].

As in any type of formal analysis, the results achieved using our method are only as good as the model on which they are based. We saw in Section 4 above that a limitation in our failure model of the Keyboard Switch led to a number of 'false positives' being produced. Whether it is worthwhile devising a more comprehensive failure model to eliminate such results, or whether the false positives can be satisfactorily explained away, depends on the security evaluator's judgement.

Similarly, an interesting aspect of the case study in Section 4 is that the scenario in which the toggle switcher suffers fault $\{2\}$, and thus connects the wrong components, is *not* highlighted as a security problem in itself, even though it would cause the operator's keystrokes to go to the wrong computer. This is because the failure model used above does not capture the operator's intent or the 'correct' behaviour of the overall device. However, we observe that this concern could be analysed, if desired, by explicitly modelling the operator's keystroke data streams, separating them into those intended for computer A and those intended for computer B. Our analysis could then be used to identify fault modes in which keystrokes intended for computer A are sent along channel B, and vice versa.

6. CONCLUSIONS

Evaluating security devices relies on the ability to assess the impact of various system failures on potential information flow. We have presented a compositional method for calculating the dependencies between a system's inputs and outputs in different fault modes. Both top-down and bottom-up evaluations were accommodated by considering dependent (global) and independent (local) fault modes, respectively.

As the examples in this paper show, composition of local fault modes results in a rapid escalation of the number of cases to be evaluated. It is for this reason that bottom-up fault evaluations, while the most 'natural' approach, because they can use well-established fault metrics for given components, have not been practical to date.

Fortunately, composition of our fault-mode dependency tables is based on simple matrix manipulations and is thus readily automatable. We are currently developing a tool which will generate the combined tables automatically, and then highlight those worrisome combinations of faults that connect high-security inputs to low-security outputs. Furthermore, the tables themselves will be generated automatically from a device's schematic diagram, since the structure of rows and columns is isomorphic to the diagram's connectivity graph. The security evaluator then merely needs to populate the tables with appropriate fault modes, which is a straightforward task, comparable to filling in truth tables for digital electronic components.

Finally, we note that there are a number of ways in which this work can be extended. For instance, we assumed above that the schematic layout of components was static, and that changes in connectivity were all due to faults occurring in individual components. However, we can readily imagine extending the system-wide analysis to model faults that change the schematic layout, e.g., to add a short circuit across the processor board, by adjusting cells in the system-wide matrices. Another issue when performing failure mode evaluations in practice is the possibility that certain security violations may arise from a particular sequence of failures, separated in time. It remains to be seen how the analysis above can be extended to handle temporal concerns.

ACKNOWLEDGEMENTS

We wish to thank Tim McComb for correcting an error in the case study, the anonymous Computer Journal reviewers for improving the presentation, Andrew Matthews and Scott Mallen for their insights into fault analysis, Luke Wildman for reviewing drafts of this paper, and Lim Soon Heng, Yeo Puay Hong and Yap Jin Hua for helping identify the implementation challenges associated with computer switching devices. This research was funded by the Defence Signals Directorate and the Australian Research Council via Linkage-Projects Grant LP0347620, *Formally-Based Security Evaluation Procedures*.

REFERENCES

[1] The Common Criteria Project Sponsoring Organisations (1999) *Common Criteria for Information Technology Security Evaluation*, 2.1 edition. ISO/IEC Standard 15408.

[2] The Common Criteria Project Sponsoring Organisations (1999) *Common Methodology for Information Technology Security Evaluation*, 1.0 edition.

[3] Barnes, J. and Praxis Critical Systems Limited (2003) *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley, Boston.

[4] Kirby, J., Jr., Archer, M., and Heitmeyer, C. (1999) Applying formal methods to an information security device: A case study. *Proceedings of the NATO Symposium on Protecting Information Systems in the 21st Century, Washington DC, USA, 25–27 October*. RTO/NATO.

[5] Ghosh, A. K., O'Connor, T., and McGraw, G. (1998) An automated approach for identifying potential

vulnerabilities in software. *Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 3–8 May*, pp. 104–114. IEEE Computer Society, Washington.

[6] Lutz, R. R. and Woodhouse, R. M. (1996) Experience report: Contributions of SFMEA to requirements analysis. *Proceedings of International Conference on Requirements Engineering, Colorado Springs, USA, 15–16 April*, pp. 44–51. IEEE Computer Society, Washington.